# Introduction Système

## Une introduction au système d'exploitation Linux

Guillaume Santini

guillaume.santini@iutv.univ-paris13.fr
IUT de Villetaneuse

2 janvier 2012

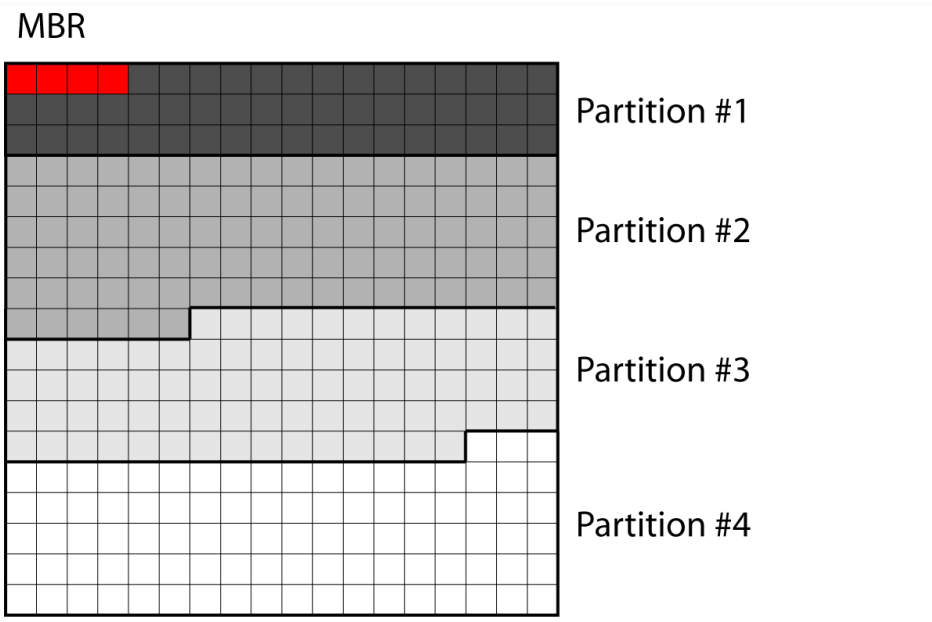
Partie #5

# Plan

# Le découpage de l'espace physique d'un disque dur

## Les blocs mémoire : le découpage élémentaire

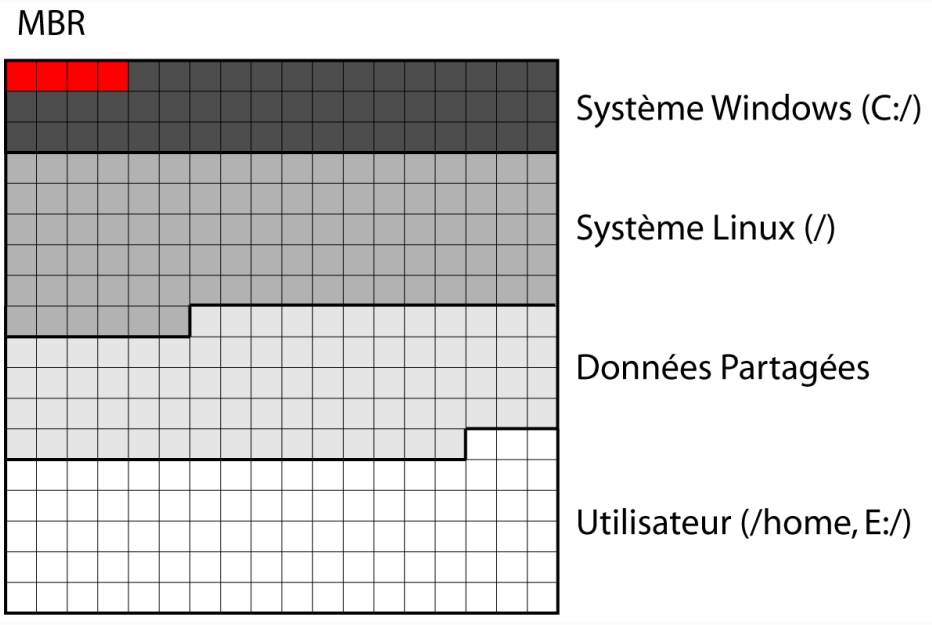
- Il s'agit d'une unité physique de stockage magnétique,
- Il est "découper" en "blocs" qui correspondent aux plus petites unités de stockage d'un système de fichier,
- La taille du bloc dépend du système de fichier.




## Les partitions : un découpage intermédiaire

- L'espace mémoire disponible d'un disque dur peut être subdivisé en plusieurs parties,
- Chaque partie peut alors accueillir un système de fichier différent.
- Cela peut permettre de
  - faire cohabiter plusieurs systèmes d'exploitation sur un même disque dur,
  - définir des zones dédiées à certains types de fichiers.

# La partition : un disque virtuel




## La partition : un disque presque comme un autre

Une partition est définie par

- Un nom,
- Une zone physique sur le disque dur,
- Un système de fichier.

## Les différents systèmes de fichier

Ils se caractérisent par :

- des tailles de blocs différents,
- des systèmes d'indexation différents permettant de retrouver l'adresse physique d'un fichier.

## Les principaux systèmes de fichier

| | | | |
|---|---|---|---|
| fat | Windows (Linux) | reiserfs | Linux |
| ntfs | Windows (Linux) | swap | Linux |
| ext | Linux | | |

# Quelques partitions spéciales

## MBR : Master Boot Record

- C'est la première partition lue lors de l'amorçage d'un ordinateur ; placée au début (physique du disque).
- Elle contient une base de registre indiquant les adresses physiques de début et de fin de chaque partitions,
- Elle permet d'indiquer où doit être lu le système d'exploitation à charger.

## SWAP dans les systèmes linux

- Elle est dédiée au stockage de l'image mémoire des processus,
- Elle se comporte comme une extension de la mémoire vive sur le disque,
- Elle permet un accès plus rapide à des données stockées temporairement et mises à jour très fréquemment sur le disque dur.

## Utiliser les partitions pour faciliter la gestion des données

- Une partition par système d'exploitation (susceptibles de changer)
- Une partition pour les programmes et applications,
- Une partition pour les données utilisateurs (conservées indépendamment des changement d'OS)
- Une partition pour les bases de données (accessible depuis plusieurs OS, plusieurs machines)

# Les principaux répertoires et leur contenu

## Une structure plus ou moins normalisée

- Les fichiers nécessaires au fonctionnement du système sont organisés en arborescence,
- Cette arborescence est commune à presque toutes les distribution linux,
- Cette organisation rationalisée facilite l'installation de nouveaux programmes qui savent où trouver les fichiers dont ils peuvent avoir besoin.

## Une organisation qui permet un cloisonnement

- Les fichiers et les répertoires systèmes sont protégés par des restrictions de droits,
- De nombreux fichiers ne peuvent être modifiés par un utilisateur "normal",
- Seul l'utilisateur `root`, ou les utilisateur faisant partie du groupe `admin` peuvent avoir la permission de modifier certains fichiers.
- Il s'agit d'une protection. Pour réaliser une action susceptible d'affecter le comportement du système il faut montrer "patte blanche" et prendre conscience de ce que l'on fait. Entrer le mot de passe `root` doit être un signal d'alerte.

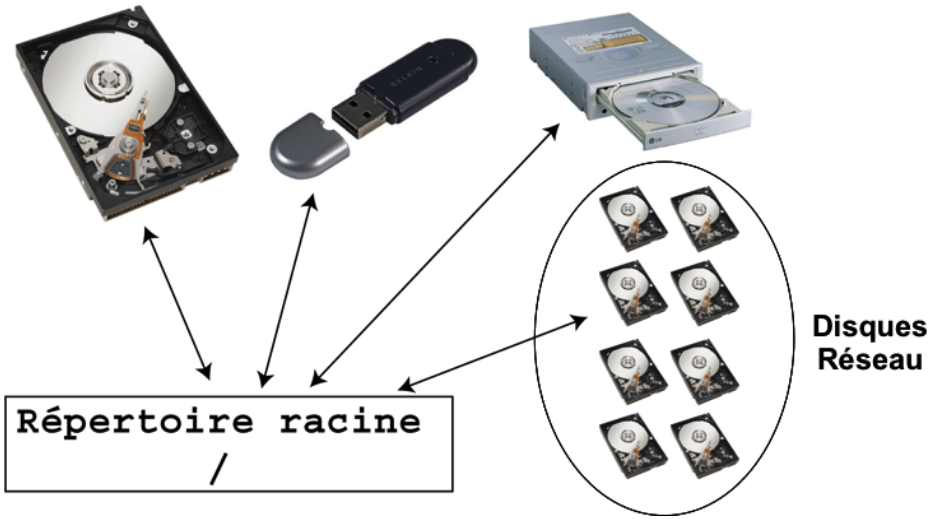
# Les principaux répertoires et leur contenu

| Répertoire | Contenu |
|---|---|
| / | Répertoire racine : Toutes les données accessibles par le système |
| /home | Les répertoires personnels des utilisateurs |
| /bin | Binaires exécutables des commandes de bases (cd, ls, mkdir, ?) |
| /lib | Librairies partagées et modules du noyeau |
| /usr | Ressources accessibles par les utilisateurs |
| /etc | Fichiers de configuration (profile, passwd,fstab... ) |
| /tmp | Données temporaires |
| /dev | Fichiers spéciaux correspondants aux périphériques |
| /mnt | Points de montage des périphériques |
| /var | Fichiers de log ou fichiers changeant fréquemment |
| /root | Répertoire personnel de l'administrateur |

# Accéder aux données stockées sur un autre disque

## Notion de disque

- Un disque est une unité de stockage physique ou virtuelle.
- Il peut s'agir d'un disque dur, d'une carte mémoire, d'une clef USB, d'une partition, ou d'un disque accessible *via* un réseau.



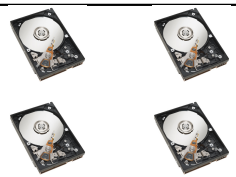

## Accès aux données depuis le système de fichier

- Chaque périphérique de stockage à un système de fichier,
- Comme toute donnée accessible est définie par un chemin partant de la racine /, les données enregistrées sur des supports périphériques doivent avoir un chemin d'accès situé dans l'arborescence.

# L'accès aux données se fait par des points de montage

## Fichiers de périphériques

Lors de l'installation du système, l'os configure des fichiers spéciaux permettant de faire le lien avec des périphériques matériels connectés sur la carte mère.

| | | |
|---|---|---|
|  | `/dev/hda1`<br>`/dev/hdb1` | `/dev/hda1`<br>`/dev/hdb1` |
|  | `/dev/sda1` | |
|  | `/dev/hdc` | |
|  | server :/share | |

## Fonctionnement

- Il s'agit d'un fichier donnant un accès à un périphérique matériel,
- Pour accéder aux données il faut "monter" le périphérique au moyen de la commande `mount`.

# mount

## Syntaxe

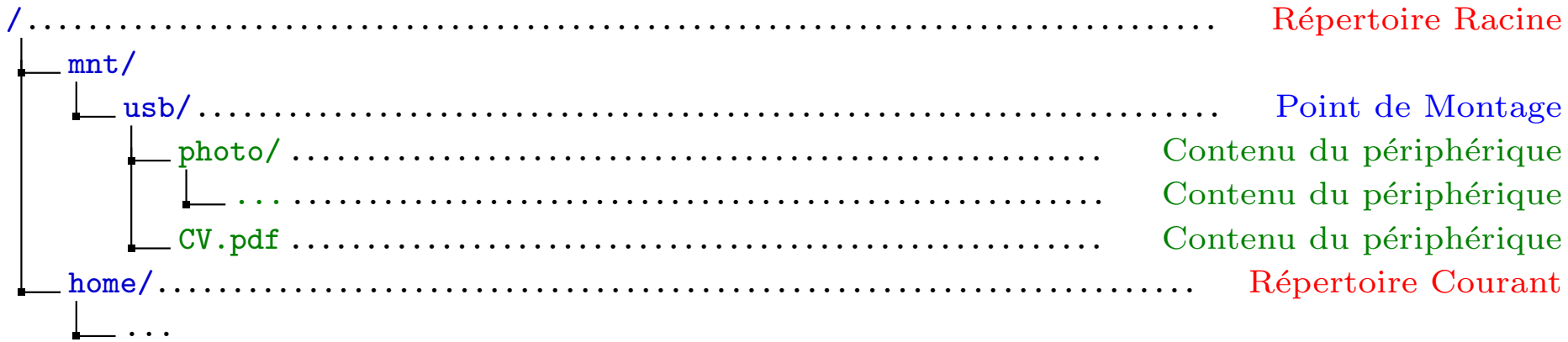
```
mount  périphérique point_de_montage
```

## Description

- `périphérique` correspond soit à un fichier de périphérique (`/dev/xxx`), soit à l'adresse d'un disque (`nom_réseau_du_disque :répertoire_du_disque`).
- `point_de_montage` correspond à un nom de répertoire valide dans l'arborescence principale donnant accès au contenu de l'arborescence du périphérique.

## Exemple d'utilisation:

```
[ login@localhost /home ] mount /dev/sda1 /mnt/usb
```



# Liste des disques montés

## Fichiers Systèmes

- /etc/fstab liste des disques à "monter" lors du démarrage du système. Il est modifiable par l'administrateur du système.
- /etc/mtab liste des disque "actuellement montés". Il est mis à jour automatiquement par le système dès qu'un disque est "monté" ou "démonté".

**/etc/fstab**

```
# /etc/fstab : static file system information.
#
# <file system> <mount point> <type> <options> <dump> <pass>
  proc          /proc             proc       defaults,noexec,nosuid          0    0
  /dev/sda1     /                 ext3       defaults,errors=remount-ro      0    1
  /dev/sda2     none              swap       sw                              0    0
  /dev/hda      /media/cdrom0     iso9660    ro,user,noauto                  0    0
  /dev/fd0      /media/floppy0    auto       rw,user,noauto                  0    0
```

**/etc/mtab**

```
/dev/sda1 / ext3 rw,errors=remount-ro 0 0
tmpfs /lib/init/rw tmpfs rw,nosuid,mode=0755 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
server-xxx :/export4/vol04/santini /users/santini nfs
rw,noatime,rsize=131072,wsize=131072,vers=3,sloppy,addr=10.10.0.191 0 0
...
```

# df

## Syntaxe

```
df -h
```

## Description

- Affiche les disques montés et leur capacité de mémoire.
- L'option `-h` (human readable) convertie l'affichage des tailles mémoires en unités conventionnelles (en nombre de blocs par défaut).

## Exemple d'utilisation:

```
[ login@localhost ~ ] df -h
  Sys. de fichiers        Taille    Uti.    Disp.    Uti%    Monté sur
  /dev/sda1               56G       16G     37G      31%     /
  myserver :/home/sant    1,8T      1,6T    192G     90%     /users/santini
  ...                     ...       ...     ...      ...     ...

[ login@localhost ~ ] ■
```

# du

## Syntaxe

```
du -sh
```

## Description

- Affiche l'espace mémoire utilisé par un fichier ou un répertoire.
- L'option `-h` (human readable) convertie l'affichage des tailles mémoires en unités conventionnelles (en nombre de blocs par défaut).
- L'option `-s` (sumurize) n'affiche pas le détail des fichiers et des sous-répertoires.

## Exemple d'utilisation:

```
[ login@localhost ~ ] du -sh Documents/
5,2G Documents/

[ login@localhost ~ ] ■
```

# Plan

# Fichiers sources → Exécutable → Processus

## Les sources : Une *"recette de cuisine"*

- Exprime un ensemble de tâches à réaliser pour accomplir le programme (le plat cuisiné).
- Utilise un langage de programmation.
- C'est un fichier texte.

## L'exécutable

- Exprime les mêmes tâches dans un langage machine.
- Ce fichier ne fonctionne que sur des ordinateurs qui ont la même architecture.
- C'est un fichier binaire.

## Les processus

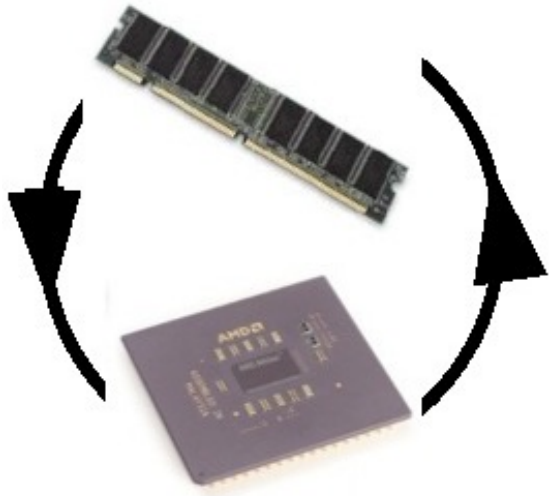
- L'évaluation des instructions machines engendre des processus.
- Ces processus sont exécutés par le matériel.
- Les instructions machine doivent donc être adaptées au matériel.

**dessine.c**

```
(...)
float r, x, y;
r=3.0;
x=0.0;
y=7.1;
cercle([0.,0.],r)
segment([0.,0.],[x,y])
```

**dessine**

```
10100101 11101001
10001001 00100101
00101010 00100010
01111011 10110101
01000010 00110011
00101101 11010100
(...)
```



# Langages Compilés Vs Langages Interprétés

## Caractéristiques des Langages Compilés

- L'ensemble du code source est compilé une seule fois avant l'exécution en instructions machine (contenues dans un fichier : exécutable).
- Le compilateur n'est pas nécessaire lors de l'exécution.
- Le compilateur est spécifique à la machine.
- L'exécutable (code compilé) est spécifique à la machine.

## Inconvenients

- Il faut recompiler pour prendre en compte une modification du code.
- L'exécutable n'est pas portable sur d'autres machines.

## Avantages

- Plus rapide (spécifique à la machine qui exécute les instructions).
- L 'ensemble des instructions sont regroupées dans un seul fichier.

## Exemples de langages Compilés

- C, C++, ADA, Pascal, Fortran, Cobol,

# Langages Compilés Vs Langages Interprétés

## Caractéristiques des Langages Interprétes

- Les instructions du code source sont converties en instructions machine lors de l'exécution du programme
- L'interpréteur est nécessaire lors de l'exécution.
- L'interpréteur est spécifique à la machine,
- L'exécutable (le code source) n'est pas spécifique à la machine.

## Inconvenients

- Moins rapide.
- Plusieurs fichiers (et librairies) servent à l'exécution.

## Avantages

- Modifications du code source immédiatement prises en compte lors de la réexécution.
- Le code est portable sur d'autres machine

## Exemples de langages Interprétés

- Java, Python, Bash, Lisp, PHP, Prolog, Perl

# Lancer un programme/une commande

## Cas général

- Pour exécuter un programme il suffit saisir sur la ligne de commande le chemin menant au fichier contenant les instructions,
- Si le fichier présente la permission `"X"` pour exécutable, les instructions qu'il contient sont exécutées.

## Cas particulier : les commandes

- Une commande (`cd`, `ls`, `python`, `firefox`, ...) est un programme comme un autre,
- Les instructions qui doivent être évaluées sont écrites dans un fichier (`/bin/cd`, `/bin/ls`, `/usr/bin/python`, `/usr/share/bin/firefox`, ...),
- Poutant ...

## Des chemins qui mènent nulle part

- les noms des commandes (`cd`, `ls`, `python`, `firefox` ...) sont toujours saisies comme des chemins relatifs (pas de `/bin/...` devant le nom du fichier), alors que le fichier de commande n'est pas dans le répertoire courant !...
- On donne donc un chemin vers un fichier qui n'existe pas ...

# Chemins par défaut et variable d'environnement

Lorsqu'on donne une commande au terminal, on ne spécifie pas le chemin vers le fichier qui contient l'exécutable, on donne juste le nom du fichier...

```
[ login@localhost ~ ] ls
Mes_Documents/ Etoiles/ astronomie.txt cv.pdf
[ login@localhost ~ ] ■
```

...alors, comment le système trouve-t-il le fichier a exécuter correspondant à la commande ?...

## Un mécanisme propre aux commandes

- Le premier mot tapé sur la ligne de commande est toujours interprétée comme le nom d'un fichier exécutable,
- Le système recherche donc dans une liste de répertoires contenant les exécutables si un fichier porte le nom de cette commande,
- Dès qu'il trouve dans ces répertoires un tel fichier, il l'exécute ...

# Chemins par défaut et variable d'environnement

## Les variables d'environnement

- Comme les variables d'un script, les variables d'environnement sont associées à une valeur,
- De telles variables sont définies par le système d'exploitation pour son fonctionnement, ce sont les variables d'environnement,
- ces variables peuvent être utilisées par les programmes.

## La variable d'environnement `$PATH`

- Sa valeur est une liste de répertoires séparés par le signe `:`.

```
PATH=/bin:/usr/bin:/usr/local/bin:/usr/X11/bin
```

- Lors de chaque appel de commande, l'interpréteur parcourt cette liste dans l'ordre à la recherche d'un fichier portant le nom de la commande,
- Dès qu'il rencontre un tel fichier, il met fin à sa recherche et exécute le fichier.

## Rôle de `$PATH`

- $\Rightarrow$ Il s'agit d'une liste de répertoires que l'interpréteur parcours automatiquement et séquentiellement (par défaut) si aucun chemin n'est donné pour trouver le fichier exécutable.

# which

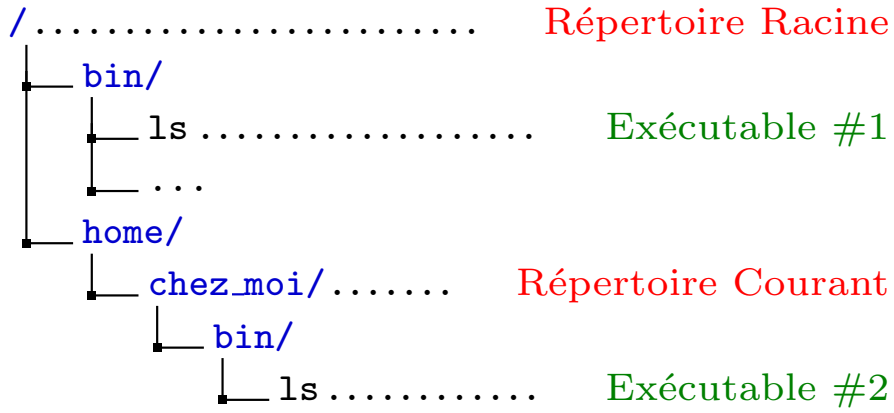
## Syntaxe

```
which  nom_de_la_commande
```

## Description

- Affiche le chemin du fichier correspondant à une commande.
- Parcours successivement les répertoires de la variable $PATH. Dès qu'il trouve un fichier correspondant au nom de la commande il renvoie son chemin.

## Exemple d'utilisation:




```
[ login@localhost /home/chez_mo ] echo $PATH
/bin:/usr/bin:/usr/local/bin:/home/chez_moi/bin

[ login@localhost /home/chez_moi ] which ls
/bin/ls
```

# Fichiers de configuration

## Fichiers systèmes et utilisateurs

- Les variables d'environnement (et d'autres variables de configuration) sont définis dans divers fichiers.
- On distingue les fichiers système qui définissent des comportements pour tous les utilisateurs (stockés dans le répertoire `/etc/`) des fichiers de configuration propres à un utilisateur (stockés dans le répertoire personnel)

| fichier | Propriétaire | Applicable à | Évalué lors |
|---|---|---|---|
| `/etc/profile` | `root` | Tous | Au début de chaque shell de login |
| `/home/chez_moi/.profile` | `utilisateur` | utilisateur | Au début de chaque shell de loginl |
| `/etc/bashrc` | `root` | Tous | Au début de chaque shell |
| `/home/chez_moi/.bashrc` | `utilisateur` | utilisateur | Au début de chaque shell |

## Configurer son environnement

- Chaque utilisateur peut redéfinir ses variables d'environnement,
- Pour cela il peut modifier le contenu des fichiers `.bashrc` et `.profile` dans son répertoire personnel,
- Ce sont des fichiers cachés (leur nom commence par un point : `.`). Pour voir si ils existent il faut utiliser l'option `-a` de la commende `ls`.

# Fichiers de configuration

## Contenu d'un fichier .bashrc

- Redéfinition des variables d'environnement,
- Définition des alias,
- Définition des fonctions,
- et de façon générale toutes les instructions que l'on souhaite évaluer lors de l'ouverture d'un nouveau shell.

**.bashrc**

```
# Mes aliases
alias ll='ls -l'
alias df='df -h'
alias rm='rm -i'
# Mes variables
PATH=$PATH:$HOME/bin
PYTHONPATH=$PYTHONPATH:$HOME/lib/python
```

## Autres variables d'environnement

`$HOME` le chemin du répertoire personnel de l'utilisateur,

`$PWD` le chemin du répertoire courant.

# alias

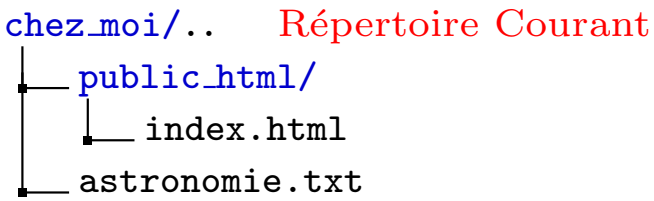
## Syntaxe

```
alias  nom_de_la_commande=expression
```

## Description

- créet un alias entre un nom de commande et une expression.
- l'expression est donnée entre *quotes : 'expression ...'*

## Exemple d'utilisation:




```
[ login@localhost ~ ] ll
-bash : ll : command not found
[ login@localhost ~ ] alias ll='ls -l'

[ login@localhost ~ ] ls -l
 total 32
drwxr-xr-x 2 santini ensinfo 4096 20 jui 15 :50     public_html
-rw-r--r-- 1 santini ensinfo   25 20 jui 15 :49 telluriques.txt
```

# dirname

## Syntaxe

dirname *chemin*

## Description

- Ne conserve que la partie répertoire d'un chemin d'accès.
- Il n'est pas nécessaire que le chemin existe dans l'arborescence. Le chemin est traité comme une chaîne de caractères.

## Exemple d'utilisation:

```
[ login@localhost ~ ] dirname Documents
.
[ login@localhost ~ ] dirname Documents/cv.txt
Documents
[ login@localhost ~ ] dirname Documents/Photos/
Documents
[ login@localhost ~ ] dirname Documents/Photos/Soleil.jpg
Documents/Photos
[ login@localhost ~ ] ■
```

# basename

## Syntaxe

basename *chemin*

## Description

- Élimine le chemin d'accès et le suffixe d'un nom de fichier.
- Il n'est pas nécessaire que le chemin existe dans l'arborescence. Le chemin est traité comme une chaîne de caractères.

## Exemple d'utilisation:

```
[ login@localhost ~ ] basename curriculum.pdf
curriculum
[ login@localhost ~ ] basename Documents/cv.txt
cv
[ login@localhost ~ ] basename Documents/Photos/Soleil.jpg
Soleil
[ login@localhost ~ ]
```

# Plan

# Les calculs arithmétiques

## Bash un langage orienté sur le traitement des chaînes de caractères

Même si ce langage n'est pas fait pour effectuer des opérations de calcul arithmétique il propose des fonctionnalités de base permettant d'effectuer des calculs simples tels que les additions, soustractions, multiplications et divisions.

## Syntaxe

`$(( expression_arithmétique ))`

## Exemples

```
[ login@localhost ~ ] total=$(( 5 + 3 ))

[ login@localhost ~ ] echo $total
8
[ login@localhost ~ ] echo $(( 5 - 3 ))
2
[ login@localhost ~ ] echo $(( 5 * 3 ))
15
[ login@localhost ~ ] echo $(( 5 / 3 ))
1
```

# for

## for Boucle itérative

- permet de répéter l'évaluation d'une ou plusieurs instructions,
- à chaque tour de boucle une variable appelée itérateur change de valeur,
- la sortie de boucle s'effectue lorsque l'itérateur atteint une certaine valeurs.

## Syntaxe #1

```
for (( init ; test ; incr )) ; do
    expr_1
    expr_2
    ...
done
```

Ici, la condition d'arrêt est sur la valeur numérique de l'itérateur.

## Exemple #1

**test_for_loop_1.bash**

```
#!/bin/bash

echo "test #1"
for (( i = 0 ; i < 3 ; i++ )) ;do
    echo '$i = '$i
done
```

```
[ login@localhost ~ ]
./test_for_loop_1.bash
test #1
$i = 0
$i = 1
$i = 2
```

# for

## for Boucle itérative

- permet de répéter l'évaluation d'une instruction,
- à chaque tour de boucle une variable appelée itérateur change de valeur,
- la sortie de boucle s'effectue lorsque l'itérateur a parcouru toute la liste.

## Syntaxe #2

```
for var in val_1 val_2 ... ; do
    expr_1
    expr_2
    ...
done
```

Ici, la boucle s'arrête lorsque toute la liste des valeurs a été parcourue.

## Exemple #2

**test_for_loop_2.bash**

```
#!/bin/bash

echo "test #2"
for i in hello la terre ;do
    echo '$i = '$i
done
```

```
[ login@localhost ~ ]
./test_for_loop_2.bash
test #2
$i = hello
$i = la
$i = terre
```

# if

## Branchements conditionnels

- Le `if` permet de mettre en place des alternatives.
- Un test (dont le résultat est Vrai ou Faux) permet de conditionner les expressions qui seront évaluées.

## Syntaxe #1

```
if test
then
        expr_1
        expr_2
        ...
fi
```

## Comportement

- Ici, les expressions ne sont évaluées que si le test renvoie la valeur Vrai.
- Aucune des expressions ne sont évaluées si le test renvoie la valeur Faux.

# if

## Syntaxe #2

```
if test
then
        expr_1
else
        expr_2
fi
```

## Comportement

- Si le test renvoie la valeur Vrai l'expression `expr_1` est évaluée, et
- sinon le test renvoie la valeur Faux c'est l'expression `expr_2` qui est évaluée.

## Syntaxe #3

```
if test_1
then
        expr_1
elif test_2
        expr_2
elif test_3
        expr_3
else
        expr_4
fi
```

## Comportement

- Si `test_1` renvoie la valeur Vrai l'expression `expr_1` est évaluée,
- si `test_2` renvoie la valeur Vrai l'expression `expr_2` est évaluée,
- si `test_3` renvoie la valeur Vrai l'expression `expr_3` est évaluée, et
- si aucun des tests ne renvoie la valeur Vrai alors c'est l'expression `expr_4` qui est évaluée.

# Les tests

## Les tests peuvent prendre plusieurs formes

Il peuvent porter sur :

- l'arborescence (présence, absence, permission sur les répertoires et fichiers),
- les chaînes de caractères,
- les valeurs numériques.

## Tests de l'arborescence

| Syntaxe | Valeur |
|---|---|
| [[ -d fichier]] | Vrai si fichier est un nom de répertoire valide (si il existe). |
| [[ -f fichier ]] | Vrai si fichier est un nom de fichier valide (si il existe). |
| [[ -r fichier ]] | Vrai si il y a le droit de lecture sur le fichier. |
| [[ -w fichier]] | Vrai si il y a le droit d'écriture sur le fichier. |
| [[ -x fichier ]] | Vrai si il y a le droit d'exécution sur le fichier. |

# Les tests

## Tests sur les chaînes de caractères

| Syntaxe | Valeur |
|---|---|
| [[ chaine_1 = chaine_2 ]] | Vrai si les 2 chaînes sont identiques. |
| [[ chaine_1 != chaine_2 ]] | Vrai si les 2 chaînes sont différentes. |
| [[ -n chaine ]] | Vrai si la chaîne est non vide. |
| [[ -z chaine ]] | Vrai si la chaîne est vide. |

## Tests sur les valeurs numériques

| Syntaxe | Valeur |
|---|---|
| [[ nb_1 -eq nb_2 ]] | Vrai si nb_1 $=$ nb_2 (eq pour equal). |
| [[ nb_1 -ne nb_2 ]] | Vrai si nb_1 $\neq$ nb_2 (ne pour not equal). |
| [[ nb_1 -gt nb_2 ]] | Vrai si nb_1 $>$ nb_2 (gt pour greater than). |
| [[ nb_1 -ge nb_2 ]] | Vrai si nb_1 $\geq$ nb_2 (ge pour greater or equal). |
| [[[ nb_1 -lt nb_2 ]] | Vrai si nb_1 $<$ nb_2 (ge pour lower than). |
| [[[ nb_1 -le nb_2 ]] | Vrai si nb_1 $\leq$ nb_2 (ge pour lower or equal). |

# Les tests

## Opérateurs booléens

| Syntaxe | Valeur |
|---|---|
| ! [[ test ]] | Vrai si le test renvoie Faux (négation). |
| [[ test_1 ]] \|\| [[ test_2 ]] | OU logique. |
| [[ test_1 ]] && [[ test_2 ]] | ET logique. |

## Tables de vérité

| **ET (&&)** | **Vrai** | **Faux** |
|---|---|---|
| **Vrai** | Vrai | Faux |
| **Faux** | Faux | Faux |

| **OU (\|\|)** | **Vrai** | **Faux** |
|---|---|---|
| **Vrai** | Vrai | Vrai |
| **Faux** | Vrai | Faux |

| **NOT (!)** | **Vrai** | **Faux** |
|---|---|---|
| | Faux | Vra |